C# Programlama

Threading, async, await

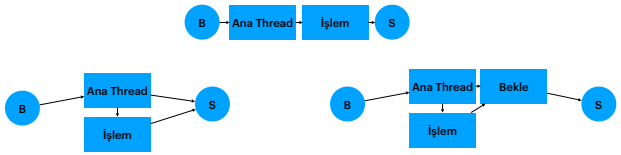
Emir Öztürk

Threading

- Process
  - Çalıştırma anındaki program
- Thread
  - En küçük parça
  - Scheduler tarafından yönetilebilir
  - <<Lightweight process>>
- Scheduler
  - Birden fazla process
  - Aynı zaman
- Multithreading

Threading

- Main Thread
- System.Threading.Thread



Threading
Parametresiz

- Senkronizasyon yok
- Parametre almaz
- Değer döndürmez

```
51
Thread sonrası
29
83
```

```
class Program {
    1 usage
    public static void Yazdir()
    {
        Console.WriteLine(new Random().Next(1,100));
        Console.WriteLine(new Random().Next(1,100));
        Console.WriteLine(new Random().Next(1,100));
    }
    public static void Main()
    {
        Thread thread = new Thread(Yazdir);
        thread.Start();
        Console.WriteLine("Thread sonrası");
    }
}
```

Threading
Parametreli

- Senkronizasyon yok
- Parametre alır
- Değer döndürmez

```
Thread sonrası
5
159
128
```

```
class Program {
    1 usage  More...
    public static void Yazdir(object sayi)
    {
        Console.WriteLine(new Random().Next(1,(int)sayi));
        Console.WriteLine(new Random().Next(1,(int)sayi));
        Console.WriteLine(new Random().Next(1,(int)sayi));
    }
    public static void Main()
    {
        Thread thread = new Thread(Yazdir);
        thread.Start(parameter:200);
        Console.WriteLine("Thread sonrası");
    }
}
```

Threading
Lambda İfadesi

- Senkronizasyon yok
- Parametre alabilir
- Değer döndürmez

```
Thread sonrası
5
```

```
public static void Main()
{
    Thread thread = new Thread(start: (x:object?)=>Console.WriteLine(new Random().Next(1,(int)x)));
    thread.Start(parameter:200);
    Console.WriteLine("Thread sonrası");
}
```

Threading
Ortak değişken kullanımı

- Birden fazla thread bir değişken
- Değer farklılığı
- Locking
- Thread.Sleep -> thread.Join()
  - Thread değişkenleri saklanmalı

```
200
200
200
200
200
200
400
400
400
400
Toplam:400
```

```
class Program
{
    private static int ortakDegisken=0;

    0 1 usage
    static void Arttir(object deger)
    {
        Thread.Sleep(millisecondsTimeout: new Random().Next(0,1000));
        int t = ortakDegisken;
        Thread.Sleep(millisecondsTimeout: new Random().Next(0,1000));
        ortakDegisken = t + (int)deger;
        Console.WriteLine(ortakDegisken);
    }
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thread thread = new Thread(Arttir);
            thread.Start(parameter: 200);
        }
        Thread.Sleep(millisecondsTimeout: 2000);
        Console.WriteLine("Toplam:"+ortakDegisken);
    }
}
```

Threading
Locking

- Object değişken
- Lock anahtar kelimesi

```
200
400
600
800
1000
1200
1400
1600
1800
2000
Toplam:2000
```

```
class Program
{
    private static int ortakDegisken=0;
    private static object kilit = false;
    0 1 usage
    static void Arttir(object deger)
    {
        lock (kilit)
        {
            Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
            int t = ortakDegisken;
            Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
            ortakDegisken = t + (int)deger;
            Console.WriteLine(ortakDegisken);
        }
    }
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thread thread = new Thread(Arttir);
            thread.Start(parameter: 200);
        }
        Thread.Sleep(millisecondsTimeout: 20000);
        Console.WriteLine("Toplam:"+ortakDegisken);
    }
}
```

Threading
Mutex

- Lock mekanizması yerine kullanılabilir

```
200
400
600
800
1000
1200
1400
1600
1800
2000
Toplam:2000
```

```
class Program
{
    private static int ortakDegisken=0;
    private static Mutex m = new Mutex();
    0 1 usage
    static void Arttir(object deger)
    {
        m.WaitOne();
        Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
        int t = ortakDegisken;
        Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
        ortakDegisken = t + (int)deger;
        Console.WriteLine(ortakDegisken);
        m.ReleaseMutex();
    }
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thread thread = new Thread(Arttir);
            thread.Start(parameter:200);
        }
        Thread.Sleep(millisecondsTimeout:20000);
        Console.WriteLine("Toplam:"+ortakDegisken);
    }
}
```

Threading
Semaphore

- Başlangıç ve maksimum thread belirler
- Senkronizasyon sağlamaz

```
200
200
200
400
400
600
600
600
800
800
Toplam:800
```

```
class Program
{
    private static int ortakDegisken=0;
    private static Semaphore s = new Semaphore(initialCount:3, maximumCount:5);
    0 usages
    static void Arttir(object deger)
    {
        s.WaitOne();
        Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
        int t = ortakDegisken;
        Thread.Sleep(millisecondsTimeout:new Random().Next(0,1000));
        ortakDegisken = t + (int)deger;
        Console.WriteLine(ortakDegisken);
        s.Release();
    }
    public static void Main()
    {
        for (int i = 0; i < 10; i++)
        {
            Thread thread = new Thread(Arttir);
            thread.Start(200);
        }
        Thread.Sleep(millisecondsTimeout:20000);
        Console.WriteLine("Toplam:"+ortakDegisken);
    }
}
```

Threading
İki thread senkronizasyonu

```
1 reference
static void Metot1()
{
    int sayac = 0;
    while (sayac++ < 10)
        Console.Write("^");
}
1 reference
static void Metot2()
{
    int sayac = 0;
    while (sayac++ < 10)
        Console.Write("|");
}

0 references
static void Main(string[] args)
{
    Thread t1 = new Thread(() => Metot1());
    Thread t2 = new Thread(() => Metot2());
    t1.Start();
    t2.Start();
}
```

```
Microsoft Visual Studio Debug Console
|^^^^^^^^^^||||||||||
```

```
static bool ilkMi = false;
1 reference
static void Metot1()
{
    int sayac = 0;
    while(sayac<10)
        if (ilkMi)
        {
            Console.Write("^");
            ilkMi = !ilkMi;
            sayac++;
        }
}
1 reference
static void Metot2()
{
    int sayac = 0;
    while(sayac<10)
        if (!ilkMi)
        {
            Console.Write("|");
            ilkMi = !ilkMi;
            sayac++;
        }
}

0 references
static void Main(string[] args)
{
    Thread t1 = new Thread(() => Metot1());
    Thread t2 = new Thread(() => Metot2());
    t1.Start();
    t2.Start();
}
```

```
Microsoft Visual Studio Debug Console
|^|^|^|^|^|^|^|^|^|^
```

Async - Await

- Senkron uygulama
- Asenkron uygulama
- Async - await
- Task sınıfı
  - Delay
  - Wait

Async - Await

Asenkron fonksiyon oluşturma

- Async
- Task
- Task.Run(()=>{})
- Await

```
private static List<int> liste = new List<int>();
0 kullanım  More...
static async void EkleGoster() {
    await Task.Run(() => {
        for (int i = 0; i < 10; i++)
        {
            liste.Add(i);
            Console.WriteLine(i);
            Task.Delay(100).Wait();
        }
    });
}
0 kullanım
static async void DummyFonksiyon() {
    await Task.Run(() => {
        for (int i = 0; i < 10; i++)
        {
            liste.Add(i);
            Console.WriteLine("Paralel "+i);
            Task.Delay(100).Wait();
        }
    });
}
```

Async - Await

Asenkron fonksiyon çağrımı

- Ana thread devam eder
- Asenkron fonksiyonlar beklenmez
- Sonuç tamamlanmadan biter

```
public static void Main()
{
    EkleGoster();
    DummyFonksiyon();
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

```
0
Toplam:0
Paralel 0
```

```
public static void Main()
{
    EkleGoster();
    DummyFonksiyon();
    Thread.Sleep(milliseconds:300);
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

```
0
Paralel 0
1
Paralel 1
2
Paralel 2
Toplam:6
3
Paralel 3
```

Async - Await

Asenkron fonksiyonların beklenmesi

- Void fonksiyon await edilemez

```
static async void EkleGoster() {
    await Task.Run(() => {
        for (int i = 0; i < 10; i++)
        {
            liste.Add(i);
            Console.WriteLine(i);
            Task.Delay(100).Wait();
        }
    });
}
```

→

```
static async Task<int> EkleGoster() {
    await Task.Run(() => {
        for (int i = 0; i < 10; i++)
        {
            liste.Add(i);
            Console.WriteLine(i);
            Task.Delay(100).Wait();
        }
    });
    return 0;
}
```

Async - Await

Asenkron fonksiyonların beklenmesi

- Main async olmalı
- Await edebilmesi için Task döndürmeli
- İlk halden farkı yok
  - Sonuç aynı

```
public static void Main()
{
    EkleGoster();
    DummyFonksiyon();
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

→

```
public static async Task Main()
{
    EkleGoster();
    DummyFonksiyon();
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

```
0
Toplam:0
Paralel 0
```

Async - Await

Asenkron fonksiyonların beklenmesi

- Await

```
public static async Task Main()
{
    EkleGoster();
    DummyFonksiyon();
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

→

```
public static async Task Main()
{
    await EkleGoster();
    DummyFonksiyon();
    Console.WriteLine("Toplam:"+liste.Sum());
}
```

```
0
1
2
3
4
5
6
7
8
9
Paralel 0
Toplam:45
```